УДК 519.6
DOI: 10.26102/2310-6018/2025.50.3.014

Влияние размера образа ключа на производительность хеш-таблиц в современных архитектурах

Е.А. Буевич✉

Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация

Резюме. Работа рассматривает способ увеличения скорости поиска в хеш-таблицах с хранением ссылки, если задача предполагает ограничение производительности пропускной способностью одного из интерфейсов между уровнями хранения (кэши L1, L2, L3, память). Для уменьшения влияния этого ограничения предлагается алгоритм оптимального использования размера кэш-линии – минимальной порции информации, передающейся между уровнями хранения. В работе показано, что существует наилучший для конкретных задачи и архитектуры размер информации о ключе в хеш-таблице (образ ключа), приведены формулы для его численного и приблизительного аналитического вычисления для случаев имеющегося и отсутствующего в таблице ключа. Рассмотрен отдельный случай использования части ключа в качестве его образа в таблице. Предложен алгоритм работы с неудобными размерами образа ключа, не являющимися степенью двойки. Приведенные результаты вычислений подтверждают увеличение производительности поиска при использовании вычисляемого размера образа ключа по сравнению с другими вариантами. Приведенный результат эксперимента подтверждает предположение, что связанное с этим усложнение кода практически не влияет на производительность из-за частичного простоя процессора. В работе подразумевается разрешение коллизий через цепочки, но схожие вычисления должны быть применимы и к другим способам с учетом их особенностей.

Ключевые слова: хеш, хеш-таблица, открытая адресация, цепочка, коллизия, параллелизм уровня памяти, кэш, кэш-линия, кэш-промах.

Для цитирования: Буевич Е.А. Влияние размера образа ключа на производительность хеш-таблиц в современных архитектурах. *Моделирование, оптимизация и информационные технологии.* 2025;13(3). URL: https://moitvivt.ru/ru/journal/pdf?id=1974 DOI: 10.26102/2310-6018/2025.50.3.014

Impact of key representation size on hash tables performance in modern CPUs

E.A. Buevich✉

Moscow State Technological University "STANKIN", Moscow, the Russian Federation

Abstract. This paper considers a method for increasing the search speed in hash tables with links if the problem assumes that the performance is limited by the throughput of one of the interfaces between the storage levels (caches L1, L2, L3, memory). To reduce the impact of this limitation, an algorithm for optimal use of the cache line size, the minimum portion of information transferred between the storage levels, is proposed. The paper shows that there is an optimal size of information about a key in a hash table (key representation) for a specific problem and architecture; equations are given for its numerical and approximate analytical calculation for the cases of a key present and absent in the table. A separate case of using a part of a key as its representation in the table is considered. An algorithm for working with inconvenient key representation sizes that are not a power of two is proposed. The presented calculation results confirm the increase in search performance when using a calculated key representation size compared to other options. The presented experimental result confirms the assumption that the associated complication of the code has virtually no effect on performance due to

partial processor idleness. The work assumes collision resolution via chains, but similar calculations should be applicable to other methods given their specific features.

Keywords: hash, hash-table, open addressing, chain, collision, memory level parallelism, cache, cache-line, cache miss.

For citation: Buevich E.A. Impact of key representation size on hash tables performance in modern CPUs. *Modeling, Optimization and Information Technology*. 2025;13(3). (In Russ.). URL: https://moitvivt.ru/ru/journal/pdf?id=1974 DOI: 10.26102/2310-6018/2025.50.3.014

Введение

Часто встречающейся проблемой при разработке программного обеспечения является получение некоторого объема данных по ключу. Алгоритм поиска должен получать некоторый ключ переменной или фиксированной длины в качестве входных данных и возвращать запись, которая содержит этот ключ или однозначно идентифицируется им. В последнем случае ключ называют первичным.

Наиболее быстрым методом поиска по первичному ключу были и остаются хеш-таблицы [1], в современных реализациях имеющие аппроксимированную сложность поиска O(1) [2]. Помимо очевидного применения их для реализации ассоциативных массивов различного масштаба и назначения (от встроенных в языки программирования структур до выделенных серверов), хеш-таблицы также широко используются в реляционных базах данных для операций объединения, группировки, удаления дубликатов. Подобные задачи обычно сопряжены с объемом данных, превышающим размер кеша процессора, и высокой их доступностью (нахождением в ОЗУ значительной части ключей, с которыми выполняется операция, к моменту ее начала) [3, 4].

Хеш-таблицы по методу разрешения коллизий разделяют на таблицы с цепочками и таблицы с открытой адресацией. В таблицах с открытой адресацией местонахождение следующего потенциально искомого элемента определяется по некоторому алгоритму, а в таблицах с цепочками каждый ключ дополнен ссылкой на следующий элемент. По этой причине хеш-таблицы с открытой адресацией имеют меньшее количество избыточной информации и меньшее количество обращений к памяти, но на практике это применимо только для таблиц с фиксированной длиной ключа и значения. Очевидно, что выделение места под максимальный размер ключа в таблице приведет к большему расходу памяти, чем хранение ссылки на область памяти, в которой ключи записаны последовательно, если разница максимального и среднего размера ключа превышает размер ссылки [1].

В работе [5] рассмотрено плотное хранение ключей переменной длины непосредственно в таблице и показано, что в этом случае поиск в таблице имеет сложность, пропорциональную отношению дисперсии длины ключа к средней его длине.

Все вышесказанное, очевидно, относится и к паре ключ+значение. Таким образом, хранение ссылки в хеш-таблице в задачах, связанных с переменной длиной ключа, часто является вынужденным шагом.

В данной работе предлагается способ увеличения производительности поиска в хеш-таблицах с хранением ссылки, в том случае если общая скорость работы ограничивается пропускной способностью памяти. В работе показано, что существует наилучший для конкретных задачи и архитектуры размер информации о ключе в хеш-таблице (далее называемой образом ключа), приведены формулы для его численного и приблизительного аналитического вычисления для случаев имеющегося и отсутствующего ключа. Также предложен алгоритм работы с неудобными размерами, не являющимися степенью двойки, и экспериментально подтверждено предположение, что связанное усложнение кода практически не влияет на производительность.

Материалы и методы

Рассмотрим задачу обслуживания некоторого количества запросов на поиск случайных ключей в хеш-таблице. Длина ключей переменная, от 1 до 64 байт. Подобная задача достаточно часто возникает при работе со словарями естественных языков или с системой доменных имен в интернете, например, DNS файлом доменной зоны.

В CPU с векторными регистрами операции хеширования и сравнения ключей занимают менее десяти тактов, в то время как операция извлечения из основной памяти – несколько сотен. В Таблице 1 приведена последовательность операций для поиска одного имеющегося в таблице ключа для реализаций с одной таблицей (цепочки, двойное хеширование, линейный поиск), максимальная длина ключа K символов, размер векторного регистра Y.

Таблица 1 – Алгоритм поиска значения по ключу в хеш-таблице
Table 1 – Algorithm for searching a value by key in a hash table

№	Операция	Число операций на ключ	Время выполнения без учета простоев, такты, T_{min}	Время выполнения с учетом простоев, такты, T_L
1	Подсчет хеша, получение номера ячейки	$\left\lceil\frac{K}{Y}\right\rceil \cdot \{L, S, 4V\}$	4	10
2	Получение образа ключа	$\{1L\}$	1	200
3	Сравнение образов ключей	$\{2V\}$	2	12
4	Если ключ не найден, идти к 2	1C	10	10
5	Получение полного ключа и сравнение	$\left\lceil\frac{K}{Y}\right\rceil \cdot \{L, V\}$	1	200
6	Если полный ключ не совпал, идти к 2	1C	1	1
7	Операция с ключом	?	2	5
	Всего		21	438

В столбце «Число операций» приведено количество операций определенного типа (L-загрузка, V-векторная операция, S-сохранение) на ключ. Количество тактов приведено для кода https://github.com/evgnort/hashtable, основанного на AVX2 и выполняемого на архитектуре Skylake, максимальный размер ключа 64 байта, средний 16 байт.

На этапе 1 хеши подсчитываются параллельно для 8 ключей за итерацию, с конца ключа, дополненного нулевыми символами (отсутствующими в алфавите ключа) до размера векторного регистра.

Для этапа 4 указана задержка в 10 тактов, исходя из равновероятности нахождения и отсутствия ключа. В том случае если один из исходов более вероятен, средняя задержка должна быть меньше из-за успешного предсказания переходов. Для этапа 6 предполагается, что ложноположительный результат достаточно редок, чтобы не учитывать неверное предсказание. В качестве времени результата взято сохранение найденного ключа в некоторой области памяти. Для этапов 2 и 5 указана примерная

задержка доступа в DRAM из предположения, что объем таблицы значительно превышает размер L3 кэша CPU.

Параллелизм уровня памяти. Для полной загрузки процессора параллельно должна выполняться обработка T_L/T_{min} ключей на одном ядре. Часто это невозможно из-за ограниченного числа одновременно возможных запросов к памяти.

```
        for (i = 0; i < steps; i++)
              {
              val0 = arr[val0]->val;
              val1 = arr[val1]->val;
              ...
#ifdef PREFETCH
              _mm_prefetch(&arr[val0]->val,PREFETCH);
              _mm_prefetch(&arr[val1]->val,PREFETCH);
              ...
#endif
              some_numeric_work();
              }
```

Рисунок 1 – Код определения уровня параллелизма памяти
Figure 1 – Memory level parallelism detection algorithm

На Рисунке 1 показан шаблон кода теста для определения возможности параллельной работы с памятью, проходящий по нескольким связным спискам (для каждого списка добавляется своя пара строк). Для того, чтобы у предвыборок было время на выполнение, в цикл добавлен некоторый объем вычислительных операций. На Рисунке 2 показан результат для процессора Intel Core I5 (Skylake).

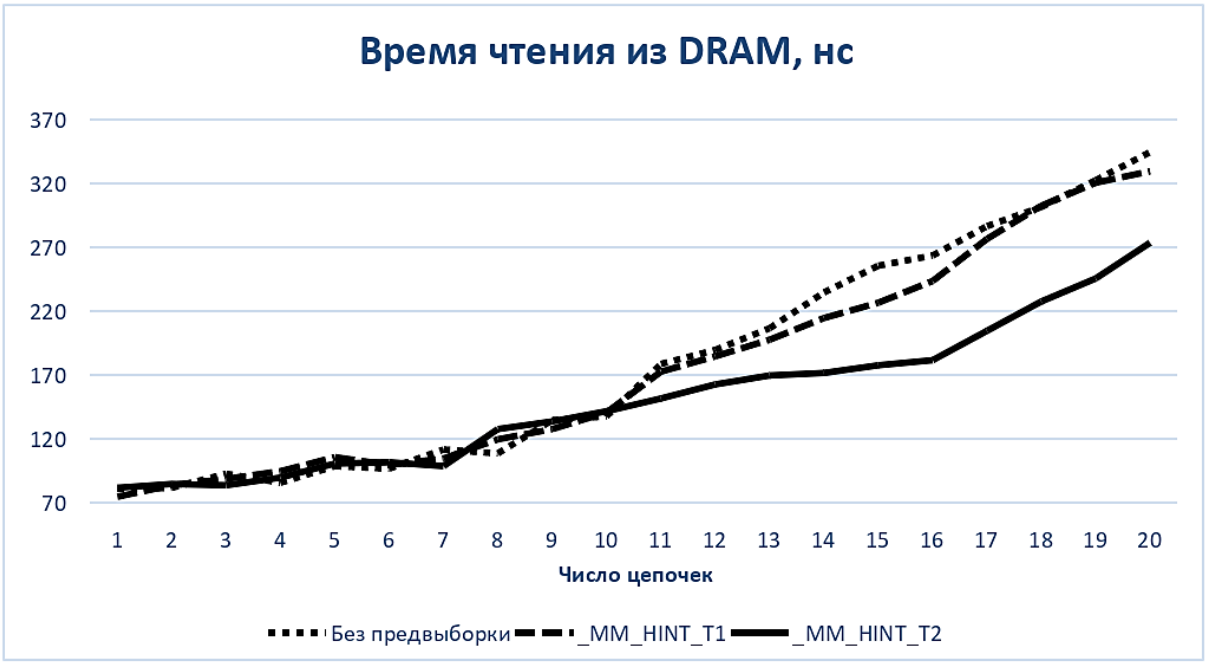

Рисунок 2 – Зависимость времени доступа к памяти от числа обращений
Figure 2 – Dependence of memory access time on the number of requests

Видно, что если целью загрузки из памяти является L1 уровень кеша, то предвыборка не оказывает видимого влияния и скорость получения данных быстро падает после 10 цепочек, что соответствует интерфейсу между L1 и L2 уровнями кеширования процессоров Intel Skylake, имеющему 10 регистров Line Fill Buffers для

получения не найденных в L1 кеш-линий. Однако, если данные предварительно помещать в L2 кеш, на графике появляется еще одна точка излома, соответствующая 16 цепочкам. Этот интерфейс также относится к вычислительному ядру и называется SuperQueue. Видно, что оба значения меньше, чем полученное для этого процессора отношение $T_L/T_{min} \approx 21$.

Также в данном процессоре существует общий интерфейс между вычислительными ядрами и L3 кешем, схожим экспериментом можно установить, что он способен выполнять одновременно 32 запроса для всех ядер, что ограничивает максимальное число параллельных потоков. В зависимости от числа задействованных ядер n, один из этих интерфейсов станет узким местом. Если обозначить максимальное количество одновременно выполняемых запросов к памяти в ограничивающем интерфейсе как I (в англоязычной литературе эта величина называется memory-level parallelism), то можно определить условие, при котором производительность реализации хеш-таблицы будет зависеть только от числа затронутых запросом кэш-линий:

$$I < n \frac{T_L}{T_{min}}. \quad (1)$$

Величину I для некоторых интерфейсов можно найти в справочниках и руководствах по процессорам, для некоторых она отсутствует в публичном доступе, может быть получена экспериментально.

Зависимость производительности системы от параметров векторных операций изучена в [6]. Векторные инструкции современных процессоров имеют достаточную ширину и производительность, чтобы условие (1) было выполнимо для простых хэш-функций (FNV[1], MurmurHash[2]), вычисляемых от слов натуральных языков или доменных имен. В качестве примера можно привести векторную реализацию хеширования кукушки в [4], приведенные результаты в которой и для AVX2 и для AVX-512 ниже максимально возможных исходя из количества инструкций и примерно соответствуют пропускной способности интерфейса между L1 и L2 кешами.

В этом случае программная предвыборка, как предлагается в [7], аппаратная предвыборка (linear probing, [1]), как и отсутствие связи по данным между запрошенными кэш-линиями [8], не влияют на производительность, поскольку тоже используют пропускную способность интерфейсов между уровнями хранения.

Блоки заголовков. DRAM может рассматриваться как внешнее блочное устройство хранения, блоком является кеш-линия. В этом случае выгодно поместить все возможные коллизии в один блок. Эта идея упомянута в [1] и широко используется в современных разработках [4, 9]. Но в этих работах размер информации о ключе в таблице выбирается исходя из условий задачи или из соображений удобства вычислений. В то же время, выполнение неравенства (1) обозначает частичный простой процессора, который можно утилизовать для работы с «неудобными» размерами образа ключа.

Рассмотрим две хеш-функции $H_1: U \to \{0,1..M\}$ и $H_2: U \to \{0,1..2^{K_s}-1\}$, определенные на пространстве ключей U. Здесь M – размер хеш-таблицы, а K_s – размер образа ключа в таблице в битах. Примем что они независимы и равномерно распределены на своей области значений. Функция H_1 определяет номер ячейки в таблице, с которого нужно начинать поиск ключа, а H_2 – образ ключа в таблице. Иногда в качестве H_2 используется некоторая часть ключа, если известно, что она относительно равномерно распределена, или CRC от ключа [6].

[1] Fowler G., Noll L., Vo K.-Ph., et al. The FNV Non-Cryptographic Hash Algorithm. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/draft-eastlake-fnv-17.html (дата обращения: 21.05.2025).
[2] tanjent. MurmurHash, final version. URL: http://tanjent.livejournal.com/756623.html (дата обращения: 21.05.2025).

Тогда вероятность ложноположительного нахождения ключа и сопутствующего кэш-промаха будет равна 2^{-K_s}, а число коллизий k для одной ячейки хеш-таблиц будет иметь вид распределения Пуассона с параметром, равным коэффициенту заполнения α [10]:

$$p(k) = \frac{e^{-\alpha}\alpha^k}{k!},$$

где $\alpha = N/M$ – коэффициент заполнения таблицы или среднее число коллизий в цепочке, N – число ключей в ней.

Число пар «заголовок ключа + значение» в одной кэш-линии ширины S_{line} будет равняться $b = \lfloor S_{line}/(S_{link} + K_s) \rfloor$, откуда $K_s = \lfloor S_{line}/b \rfloor - S_{link}$.

Число кэш-промахов при поиске имеющегося ключа с номером m в цепочке будет выглядеть как

$$v(b,m) = (m-1)2^{S_{link} - S_{line}/b} + \left\lceil \frac{m}{b} \right\rceil,$$

тогда для всех ключей в цепочке из k коллизий общее число кэш-промахов будет равно:

$$V(b,k) = \sum_{i=1}^{k} v(b,k) = 2^{S_{link} - \frac{S_{line}}{b}} \sum_{i=1}^{k-1} i + \sum_{i=1}^{k} \left\lceil \frac{i}{b} \right\rceil = 2^{S_{link} - 1 - S_{line}/b} k(k-1) + \\ + \sum_{i=1}^{k} \left\lceil \frac{i}{b} \right\rceil,$$

откуда среднее число промахов для всех ключей в таблице составит:

$$W(b) = \frac{M \cdot \sum_{j=1}^{\infty} p(j,\alpha) \cdot V(b,j)}{\sum_{j=1}^{\infty} p(j,\alpha) \cdot j} = \frac{1}{\alpha} \sum_{j=0}^{\infty} p(j,\alpha) \cdot V(b,j),$$

$$W(b) = \frac{2^{S_{link} - S_{line}/b}}{2a} \sum_{j=0}^{\infty} \frac{e^{-\alpha}\alpha^j}{j!} \cdot j(j-1) + \frac{1}{\alpha} \sum_{j=0}^{\infty} \left(\frac{e^{-\alpha}\alpha^j}{j!} \sum_{i=1}^{j} \left\lceil \frac{i}{b} \right\rceil \right),$$

$$W(b) = 2^{S_{link} - S_{line}/b} \frac{\alpha}{2} + \frac{1}{\alpha} \sum_{j=0}^{\infty} \left(\frac{e^{-\alpha}\alpha^j}{j!} \sum_{i=0}^{j} \left\lceil \frac{i}{b} \right\rceil \right). \quad (2)$$

Нужно отметить, что с изменением количества ключей в кеш-линии реальный коэффициент заполнения, отражающий количество занятых мест под ключ, также будет меняться, в то время как среднее число коллизий остается неизменным. Для учета этого примем $\alpha = \gamma b$, где γ – реальный коэффициент заполнения. Тогда среднее число кеш-промахов будет:

$$W(b) = 2^{S_{link} - S_{line}/b} \frac{\gamma b}{2} + \frac{1}{\gamma b} \sum_{j=0}^{\infty} \left(\frac{e^{-\gamma b}(\gamma b)^j}{j!} \sum_{i=0}^{j} \left\lceil \frac{i}{b} \right\rceil \right). \quad (3)$$

На Рисунке 3A показана хеш-таблица, описываемая формулой (2), а на Рисунке 3B – формулой (3). В случае 3A избыточной информацией являются только ссылки, но должно существовать преобразование ${H_2}^{-1}: \{0..2^{K_s} - 1\} \rightarrow$ часть ключа. В случае 3B такого ограничения нет, но избыточной информацией является вся хеш-таблица. Штриховкой отдельно для каждого случая показаны области памяти одинакового размера.

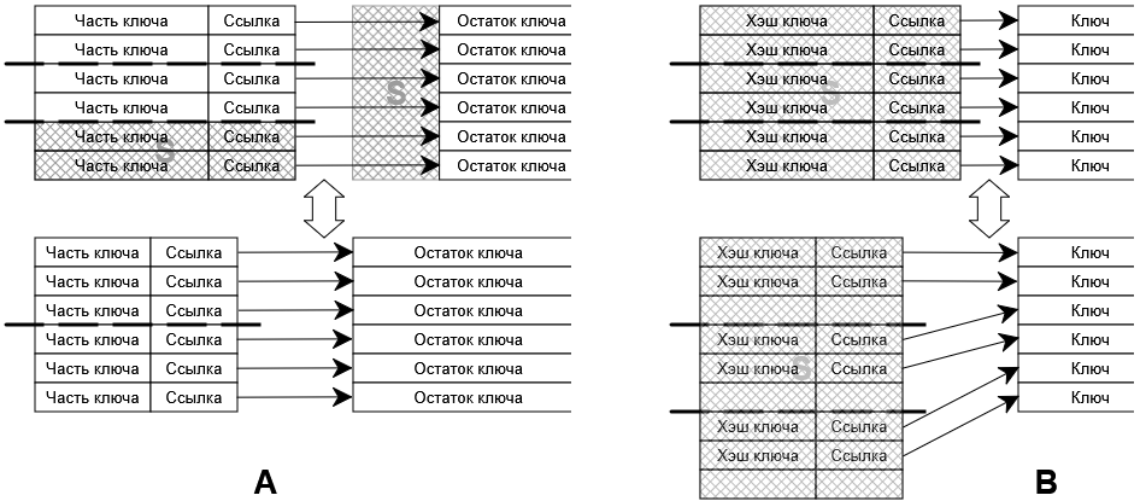

Рисунок 3 – Возможные способы организации хеш-таблицы со ссылками
Figure 3 – Possible ways of organizing a hash table with links

Для 3В можно найти приблизительное аналитическое выражение для экстремума (2). Заменим $\left\lceil \frac{i}{b} \right\rceil$ на $\frac{i}{b} + \varepsilon$, где $\varepsilon = \frac{b-1}{2b}$ – средняя ошибка количества кеш-промахов, получаемых от длины цепочки:

$$\widetilde{W}(b) = 2^{S_{link} - {S_{line}}/{b}} \frac{\alpha}{2} + \frac{1}{\alpha} \sum_{j=0}^{\infty} \left(\frac{e^{-\alpha} \alpha^j}{j!} \sum_{i=1}^{j} \left(\frac{i}{b} + \frac{b-1}{2b} \right) \right),$$

$$\widetilde{W}(b) = 2^{S_{link} - {S_{line}}/{b}} \frac{\alpha}{2} + \frac{1}{\alpha} \sum_{j=0}^{\infty} \left(\frac{e^{-\alpha} \alpha^j}{j!} \left(j \frac{b-1}{2b} + \sum_{i=1}^{j} \frac{i}{b} \right) \right),$$

$$\widetilde{W}(b) = 2^{S_{link} - {S_{line}}/{b}} \frac{\alpha}{2} + \frac{1}{2b} + \frac{1}{2}.$$

$\widetilde{W}(b)$ имеет единственный минимум:

$$\frac{d\widetilde{W}(b)}{db} = \frac{\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link} - {S_{line}}/{b} - 1}}{2b^2},$$

$$\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}} \cdot 2^{-{S_{line}}/{b}} = 1,$$

$$b = \frac{S_{line}}{\log_2 \left(\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}} \right)} = \frac{S_{line} * \ln 2}{ln \left(\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}} \right)}. \quad (4)$$

По физическому смыслу $S_{line} \in \mathbb{Z}_+$, $S_{link} \in \mathbb{Z}_+$, $\alpha \in \mathbb{R}$, $\alpha \geq 0$. В этой области решение не существует только при $\alpha = 0$, поскольку в этом случае все решения равнозначны.

При поиске отсутствующего ключа удобнее отталкиваться от числа коллизий. Число промахов при поиске в цепочке длиной k равно:

$$v(b,k) = k \cdot 2^{S_{link} - S_{line}/b} + \left\lceil \frac{k}{b} \right\rceil,$$

а при поиске в пустой цепочке равно единице. Тогда среднее число промахов при поиске отсутствующего ключа в таблице равно:

$$W(b) = e^{-\alpha} + \sum_{j=0}^{\infty} \frac{e^{-\alpha} \alpha^j}{j!} \cdot \left(j \cdot 2^{S_{link} - S_{line}/b} + \left\lceil \frac{j}{b} \right\rceil \right),$$

$$W(b) = e^{-\alpha} + 2^{S_{link} - S_{line}/b} \alpha + \sum_{j=0}^{\infty} \frac{e^{-\alpha} \alpha^j}{j!} \cdot \left\lceil \frac{j}{b} \right\rceil \quad (5)$$

и если принять $\alpha = \gamma b$, то

$$W(b) = e^{-\gamma b} + 2^{S_{link}-S_{line}/b}\gamma b + \sum_{j=0}^{\infty} \frac{e^{-\gamma b}(\gamma b)^j}{j!} \cdot \left\lceil \frac{j}{b} \right\rceil. \quad (6)$$

Аналогично можно получить приблизительную формулу для экстремума (5):

$$W(b) = e^{-\alpha} + 2^{S_{link}-S_{line}/b}\alpha + \sum_{j=0}^{\infty} \frac{e^{-\alpha}\alpha^j}{j!} \cdot \left(\frac{j}{b} + \frac{b-1}{2b}\right),$$

$$W(b) = e^{-\alpha} + 2^{S_{link}-S_{line}/b}\alpha + \frac{\alpha}{b} + \frac{1}{2} - \frac{1}{2b},$$

$$\frac{dW(b)}{db} = 2^{S_{link}}\alpha \cdot 2^{-\frac{S_{line}}{b}} \cdot \ln 2 \cdot \frac{S_{line}}{b^2} - \frac{\alpha}{b^2} + \frac{1}{2b^2},$$

$$2^{S_{line}/b} = \frac{a \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}}}{\alpha - \frac{1}{2}},$$

$$b = \frac{S_{line} \cdot \ln 2}{\ln\left(\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}}\right) - ln(\alpha - \frac{1}{2})}. \quad (7)$$

Уравнение (7) не имеет решения при $\alpha \leq 0{,}5$, однако практического интереса эта область не имеет, а также при $\alpha \cdot S_{line} \cdot \ln 2 \cdot 2^{S_{link}} = \alpha - \frac{1}{2}$, что невозможно по физическому смыслу S_{line} и S_{link}.

Результаты

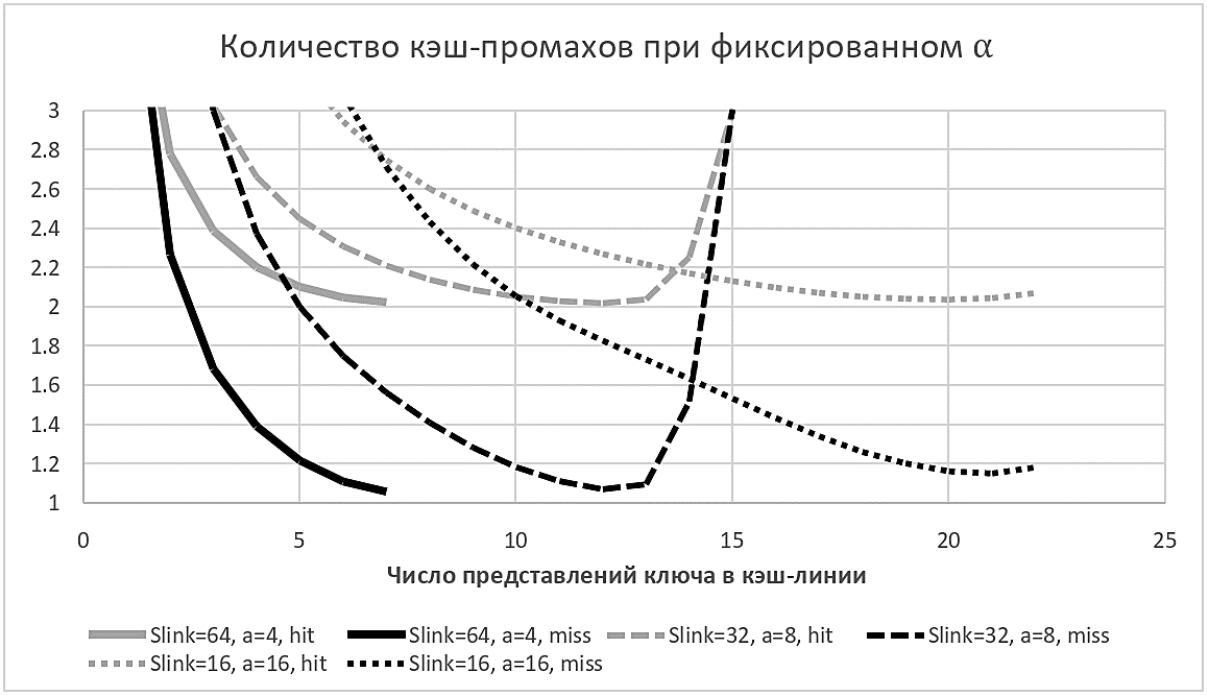

Рисунок 4 – Количество кэш-промахов при фиксированном α
Figure 4 – Number of cache misses at a fixed α

На Рисунке 4 показано среднее количество кеш-промахов при поиске отсутствующего и имеющегося ключа, рассчитанное по формулам (2) и (5) для размеров ссылок 16, 32 и 64 бита. α выбиралось как соответствующее коэффициенту заполнения 1 для «удобного» количества образов в кеш-линии (размер образа ключа равен размеру ссылки). Из графиков видно, что переход от удобного к оптимальному размеру образа

дает выигрыш во всех рассмотренных случаях от ~3 % (имеющиеся ключи, размер ссылки 16 бит) до ~31 % (отсутствующие ключи, размер ссылки 32 бита).

Таблица 2 – Оптимальное количество образов ключей в кеш-линии
Table 2 – Optimal number of key representations in a cache-line

Размер значения	Имеющиеся ключи, формула (2)	Имеющиеся ключи, формула (4)	Отсутствующие ключи, формула (5)	Отсутствующие ключи, формула (7)
64, $\alpha = 4$	7	6,87	7	7,04
32, $\alpha = 8$	12	11,77	12	12,62
16, $\alpha = 16$	20	17,9	21	20,88

В Таблице 2 приведено количество ключей, соответствующее минимумам функций (2) и (5) и соответствующее количество, полученное по формулам (4) и (7). Точность оценки падает с ростом числа коллизий, однако может дать стартовую точку для численного поиска. Полезным свойством является близость минимумов для имеющихся и отсутствующих ключей.

Иначе выглядят результаты для фиксированного γ, приведенные на Рисунках 5 и 6. Для присутствующего в таблице ключа в рассматриваемом диапазоне имеется также единственный минимум, дающий незначительный выигрыш в сравнении с «удобным» вариантом реализации. Но для отсутствующего ключа при $\gamma \geq 1$ соответствующий минимум пропадает, и функция становится неубывающей в рассматриваемой области определения. Таким образом, для таблиц, соответствующих Рисунку 3А, оправданность подобной оптимизации зависит от условий задачи.

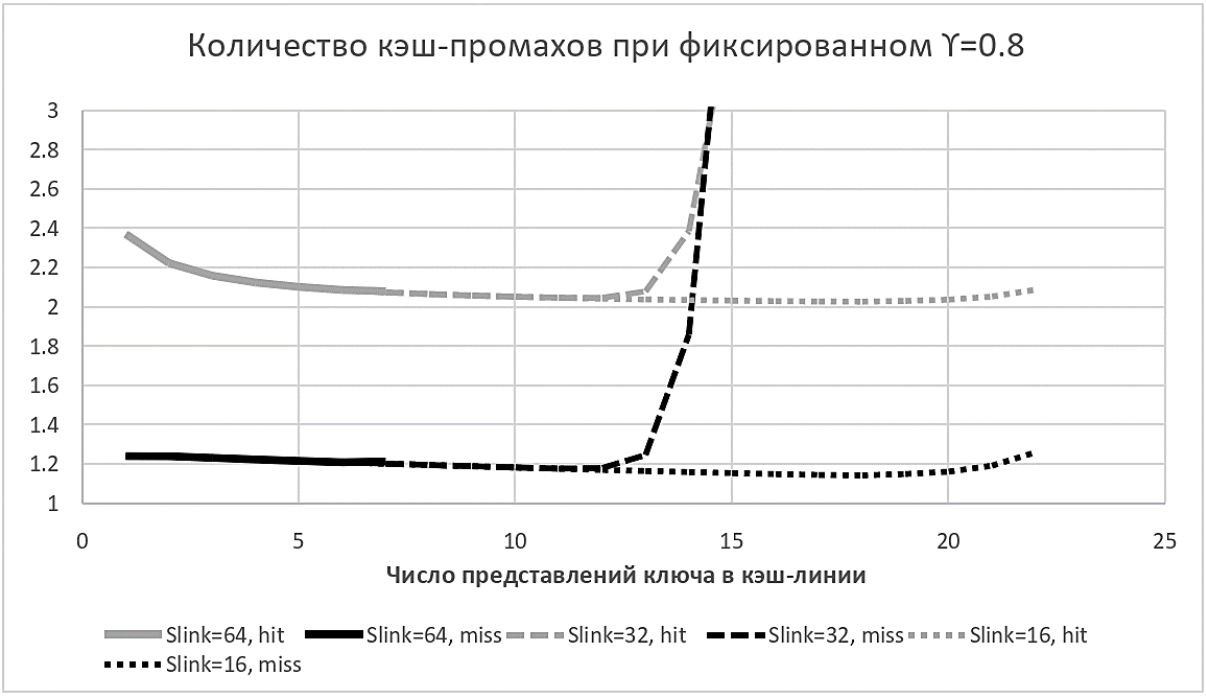

Рисунок 5 – Количество кэш-промахов при фиксированном $\gamma = 0{,}8$
Figure 5 – Number of cache misses at a fixed $\gamma = 0.8$

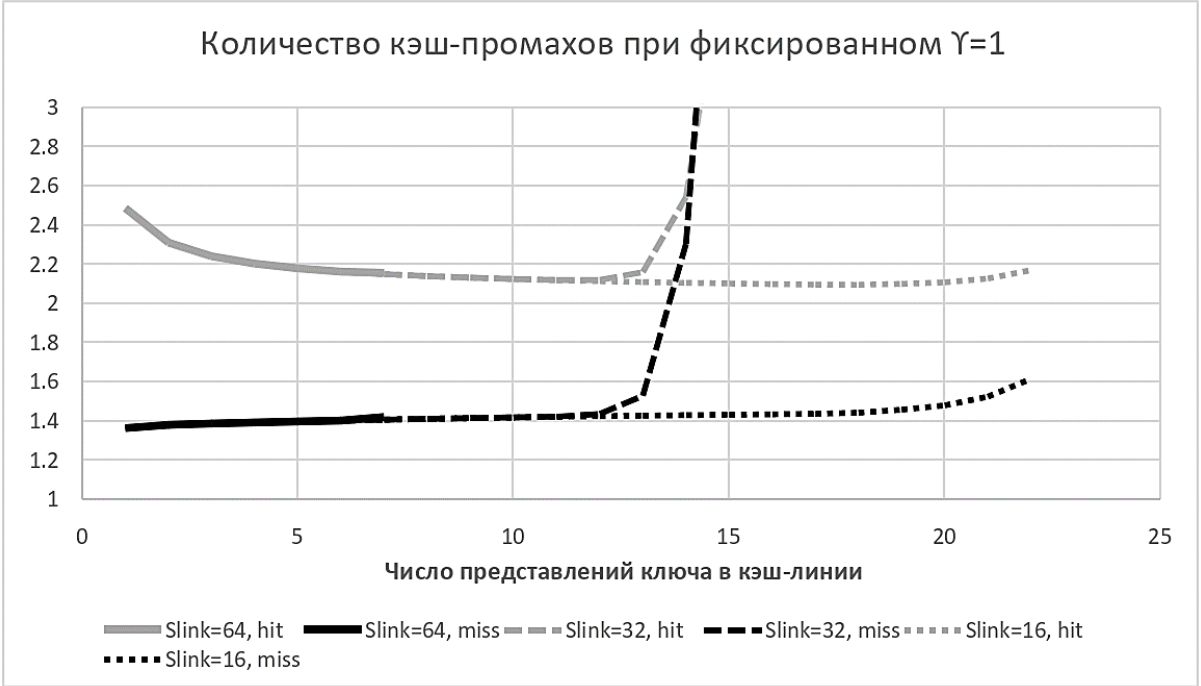

Рисунок 6 – Количество кэш-промахов при фиксированном $\gamma = 1$
Figure 6 – Number of cache misses at a fixed $\gamma = 1$

Таблица 3 – Время поиска ключа в различных реализациях
Table 3 – Key search time in different implementations

	8 образов в линии	12 образов в линии	Отношение
Теоретическое число кэш-промахов для имеющегося ключа	2,140382	2,020135	1,059
Теоретическое число кэш-промахов для отсутствующего ключа	1,411172	1,071611	1,316
Время поиска имеющегося ключа, нс	52,546	49,325	1,065
Время поиска отсутствующего ключа, нс	36,249	28,063	1,291

В Таблице 3 приведены результаты практического эксперимента. Код, доступный по ссылке https://github.com/evgnort/hashtable, реализует поиск 1 миллиона ключей, соответствующих доменным именам второго уровня, в хеш-таблице, имеющей 125 тысяч ячеек, что соответствует фиксированному $\alpha = 8$.

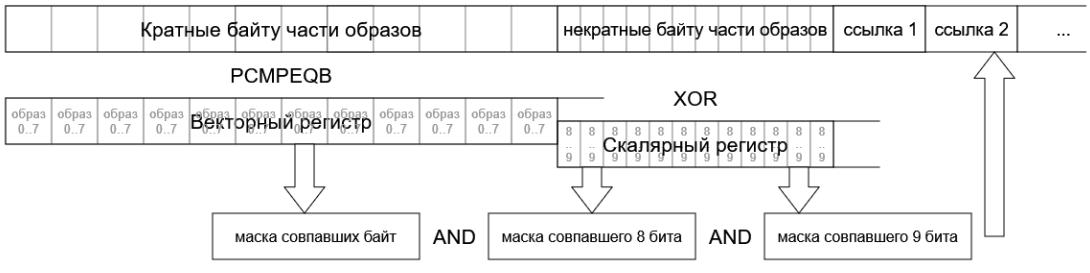

Рисунок 7 – Векторный поиск образа с некратным байту числом бит
Figure 7 – Vector search for an image with a number of bits that is not a multiple of a byte

В данном случае оптимальное количество образов ключей в кэш-линии равно 12. Для векторного сравнения упакованного некратного восьми числа бит кэш-линия организована следующим образом: в первых 12 байтах хранится по 8 бит от каждого образа, далее в 4-х байтах хранится дополнительно по 2 бита, далее хранится 12 ссылок (Рисунок 7). Векторной операцией выполняется сравнение части, кратной байту, несколькими скалярными – сравнения с каждым битом остатка. Логическое умножение результатов даст маску ссылок на ключи, образ которых совпадает с искомым.

Замеры выполнялись на процессоре Intel Core I5 (Skylake). Из результатов видно, что соотношение времени поиска ключей для удобной и оптимальной реализации отклоняется от соотношения теоретически полученного числа кэш-промахов не более чем на 2 %, в случае отсутствующих, и на 0,6 % в случае имеющихся в таблице ключей. Это подтверждает предположение о том, что производительность данной таблицы определяется только числом кэш-промахов.

Также интерес представляет изменение размера ссылки. Очевидно, что размер пространства хранения ключей и значений зависит от количества ключей, и если эта зависимость имеет предсказуемый характер, то разрядность ссылки можно привести в соответствие некоторому значению α, после которого будет выполняться перестроение таблицы. В Таблице 4 показано количество кэш-промахов для $\alpha = 8$ при изменении размера ссылки от 24-х до 32-х бит, что дает дополнительно 1–5 % производительности.

Таблица 4 – Зависимость числа кэш-промахов от размера ссылки
Table 4 – Dependence of the number of cache misses on the link size

Размер ссылки	Количество промахов, ключ есть	Количество промахов, ключа нет
24	2,005	1,019
25	2,006	1,021
26	2,008	1,025
27	2,009	1,036
28	2,010	1,038
29	2,012	1,042
30	2,016	1,050
31	2,018	1,068
32	2,020	1,072

Заключение

Приведенные результаты практического эксперимента подтверждают гипотезу о том, что для хеш-таблиц, значительно превышающих размер кэша процессора, скорость работы любого алгоритма разрешения коллизий определяется количеством кэш-линий, затронутых работой алгоритма.

Факторами, влияющими на скорость поиска ключа, являются, с одной стороны, число ключей, попарно дающих коллизию, информацию о которых возможно собрать в одной кеш-линии, а с другой – объем этой информации, снижающий вероятность ложноположительного нахождения ключа и соответственно запроса дополнительных кеш-линий.

В случае, если образ используется для хранения информации, экстремум этой функции для имеющегося в таблице ключа зависит от размера кэш-линии, размера ссылки и коэффициента заполнения, определяется согласно уравнению (3). Для отсутствующего ключа, согласно уравнению (6), оптимальным является хранение одного образа на линию при коэффициенте заполнения больше или равном единице.

В случае если образ используется только для поиска, к перечисленным факторам добавляется уменьшение реального коэффициента заполнения при росте числа образов ключей в кеш-линии, что приводит к увеличению производительности при поиске как отсутствующих, так и имеющихся ключей. Оптимальный размер образа ключа в этом случае может быть приблизительно получен аналитически из уравнений (4) и (7).

Размер хранимой ссылки также влияет на производительность. Если размер пары ключ+значение ограничен, то разрядность ссылки на них естественным образом зависит от числа ключей в таблице, что дает дополнительную возможность для оптимизации быстродействия.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Кнут Д.Э. *Искусство программирования. Том 3. Сортировка и поиск.* Москва: И.Д. Вильямс; 2018. 832 с.
   Knuth D.E. *The Art of Computer Programming. Volume 3. Sorting and Searching.* Moscow: I.D. Vil'yams; 2018. 832 p. (In Russ.).
2. Farach-Colton M., Krapivin A., Kuszmaul W. Optimal Bounds for Open Addressing Without Reordering. In: *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS), 27–30 October 2024, Chicago, IL, USA.* IEEE; 2024. P. 594–605. https://doi.org/10.1109/FOCS61266.2024.00045
3. Zukowski M., Héman S., Boncz P. Architecture-Conscious Hashing. In: *DaMoN '06: Proceedings of the 2nd International Workshop on Data Management on New Hardware, 25 June 2006, Chicago, IL, USA.* New York: Association for Computing Machinery; 2006. https://doi.org/10.1145/1140402.114041
4. De Cristo F., De Almeida E., Alves M. ViViD Cuckoo Hash: Fast Cuckoo Table Building in SIMD. In: *Proceedings of the 20th Symposium on High Performance Computing Systems, SSCAD 2019, 16–18 October 2019, Campo Grande, MS, Brasil.* Porto Alegre: Sociedade Brasileira de Computação; 2019. P. 288–299. https://doi.org/10.5753/wscad.2019.8676
5. Thorup M. String Hashing for Linear Probing. In: *SODA '09: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, 04–06 January 2009, New York, USA.* Philadelphia: Society for Industrial and Applied Mathematics; 2009. P. 655–664. https://doi.org/10.1137/1.9781611973068.72
6. Barredo A., Cebrian J.M., Valero M., Casas M., Moreto M. Efficiency Analysis of Modern Vector Architectures: Vector ALU Sizes, Core Counts and Clock Frequencies. *The Journal of Supercomputing.* 2020;76(3):1960–1979. https://doi.org/10.1007/s11227-019-02841-6
7. Chen S., Ailamaki A., Gibbons P.B., Mowry T.C. Improving Hash Join Performance Through Prefetching. *ACM Transactions on Database Systems.* 2007;32(3). https://doi.org/10.1145/1272743.1272747
8. Ross K.A. Efficient Hash Probes on Modern Processors. In: *2007 IEEE 23rd International Conference on Data Engineering, 15 April 2006 – 20 April 2007, Istanbul, Turkey*. IEEE; 2007. P. 1297–1301. https://doi.org/10.1109/ICDE.2007.368997
9. Herlihy M., Shavit N., Tzafrir M. Hopscotch Hashing. In: *Distributed Computing: 22nd International Symposium, DISC 2008: Proceedings, 22–24 September 2008, Arcachon, France.* Berlin, Heidelberg: Springer; 2008. P. 350–364. https://doi.org/10.1007/978-3-540-87779-0_24
10. Tripathi R.R.K., Singh P.K., Singh S. Templing and Temple Search-Same Height Hashing. [Preprint]. ResearchSquare. URL: https://doi.org/10.21203/rs.3.rs-1947527/v1 [Accessed 21st May 2025].

ИНФОРМАЦИЯ ОБ АВТОРЕ / INFORMATION ABOUT THE AUTHOR

Буевич Евгений Андреевич, аспирант, Московский государственный технологический университет «СТАНКИН», Москва, Российская Федерация.
e-mail: gftcompmod@gmail.com

Evgeniy A. Buevich, Postgraduate, Moscow State Technological University "STANKIN", Moscow, the Russian Federation.

Статья поступила в редакцию 27.05.2025; одобрена после рецензирования 27.06.2025; принята к публикации 08.07.2025.

The article was submitted 27.05.2025; approved after reviewing 27.06.2025; accepted for publication 08.07.2025.